CS489 Report: Flute Synthesis

Dave Pagurek (dpagurek, 20551040)

April 2019

Contents

Motivation

I record music as a hobby, and the music I produce occasionally includes recordings of my own flute playing. I do this because I can get a level of expressivity on the flute that I can't achieve as easily with the synthesizers I work with. I set out to construct a synthesizer that lets me imitate that sound, including the ability to control the intensity, vibrato, breathiness, and other parameters that I make use of when playing a real instrument. I hoped to create a tool that would give me more flexibility when prototyping new pieces, both in terms of the ease with which I can tweak and experiment and also the freedom of not having to disturb roommates should I decide to work at inconvenient times of the day.

My goal was both to produce a flute synthesizer, but also to gain a better understanding of the underlying processes making the flute sound like a flute. So, to begin with, I turned to theory and data to model the harmonics present in the sound of the flute.

Flute harmonics

The theory

The flute can be modelled as an open air column [1] for the purposes of examining its resonance. Resonant frequencies arise in open air columns due to the standing pressure wave patterns that are able to form. The geometry of the column allows standing waves with integer number of nodes, shown in Figure 1.

Each standing wave has a wavelength relative to half the length of the instrument [2]. The frequency of the note produced by each wave is found by the equation $f = \frac{v}{\lambda}$, where v is the speed of sound, and λ is the wavelength. This tells us that the lowest resonant frequency is proportional to twice the length of the instrument. Each successively higher frequency increases

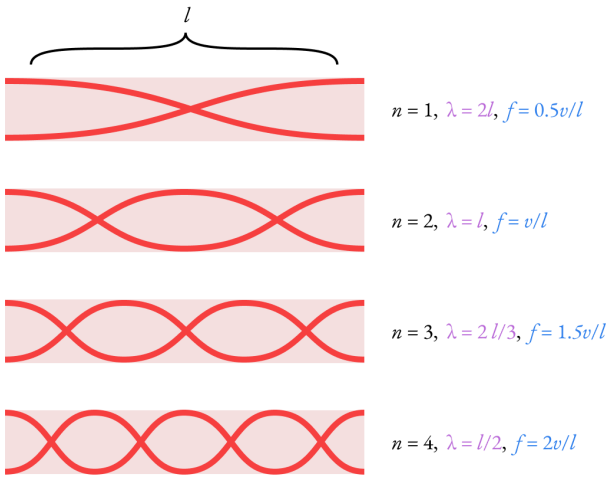


Figure 1: The frequency of a harmonic relative to the number of nodes. Each successive harmonic has a frequency which is an integer multiple of the lowest note that can be sounded.

by a factor of half the length of the instrument, and every harmonic above that is an integer multiple of this lowest frequency.

Given that the length of a flute is around 66 cm, we can estimate the lowest note, corresponding to one node:

$$\begin{aligned} f &= \frac{v}{\lambda} \\ &= \frac{340.27 \text{ m/s}}{2 \cdot 66 \text{ cm}} \quad = 257.78 \text{ Hz} \end{aligned}$$

This frequency corresponds to the note B3 and 74 cents. This agrees with the experience of playing the flute, where the lowest note that can be fingered is C3 by covering every hole. If we continue adding doubling the frequency, we start filling in all the possible harmonics. Table 1 shows what notes these correspond to.[1]

Table 1: Harmonics of C_3

Harmonic	Frequency	Note
1	261.6256	C3
2	523.2511	C4
3	784.8767	G4 and 2 cents
4	1046.5023	C5
5	1308.1278	D#5 and 86 cents
6	1569.7534	G5 and 2 cents
7	1831.3790	A6 and 69 cents
8	2093.0045	C6
9	2354.6301	D6 and 4 cents
10	2616.2557	D#6 and 86 cents

[1]It is interesting to note here that the harmonic frequencies are not in tune. We perceive a doubling of a frequency to be a jump up an octave, so octaves are found to be proportional to 2^n for increasing n. Equal-tempered tuning divides the space between octaves into 12 equally spaced semitones, so semitones are proportional to $2^{\frac{n}{12}}$. Integer multiples of a base note do not always align with these twelfths. Some instruments purposefully detune notes that are intended to be played in a chord with other notes so that the harmonics do not interfere dissonantly with the rest of the chord [3].)

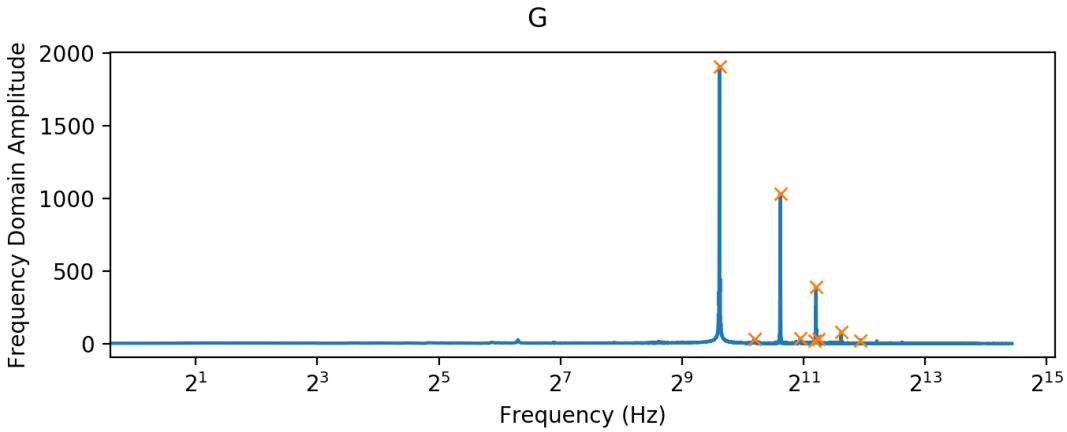
These are the frequency ratios that will have to be simulated in order to create a convincing flute synthesizer. Higher notes are made by uncovering holes on the flute to effectively shorten its length and increase the frequency of the lowest note that can be sounded.

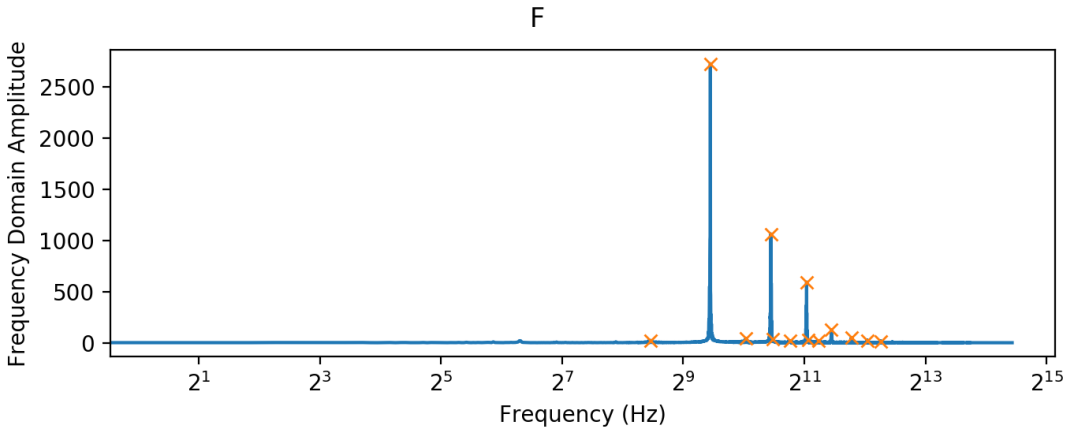
The next piece of information required is the amplitude of each harmonic. This is what will make a flute sound like a flute. For that, I turn to data from actual recordings.

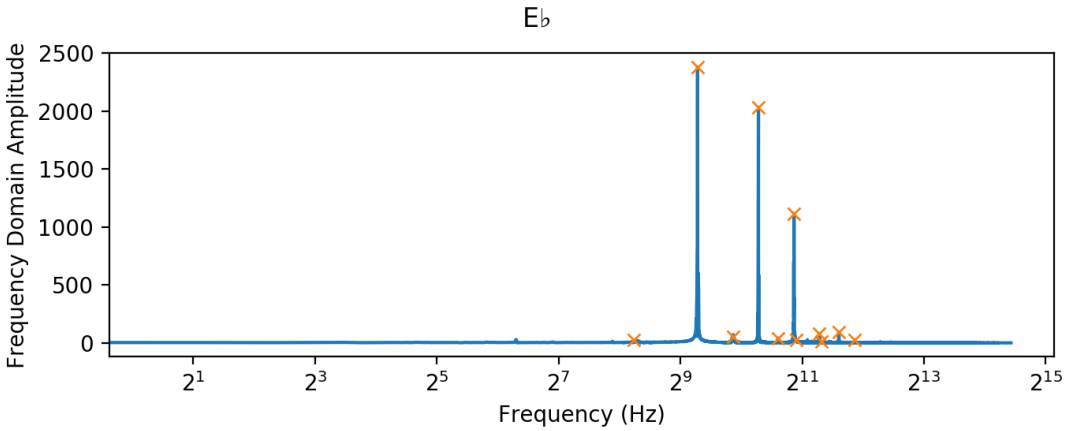
Harmonics in the wild

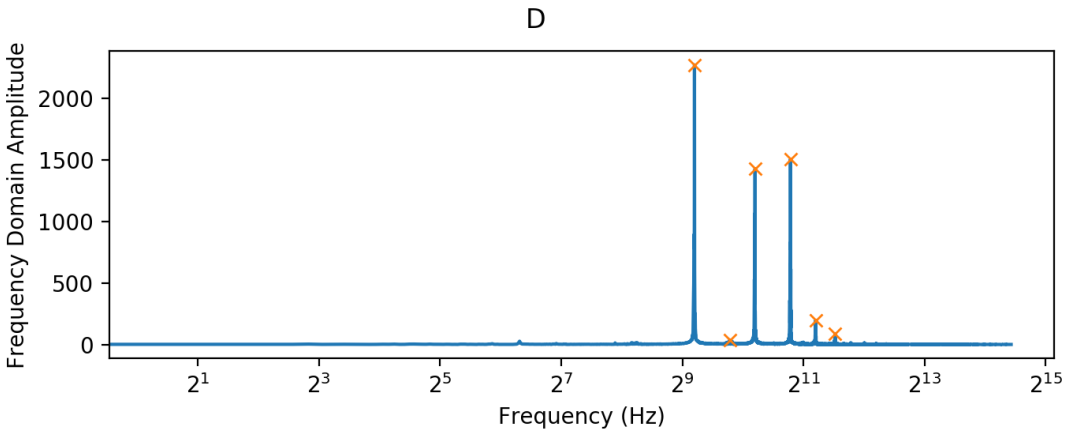
In 2018, I recorded a song [4] that featured recordings of myself playing the flute. As a starting point, I extracted four samples from one flute track in the song where a single note is held for around a second. There are samples for the notes G, F, E♭, and D. Each file is a mono wav file at a 44100 Hz, cut down to just be the steady-state portion of the held note. I processed each sample to get a sense of the harmonics present in the sound of the flute.I processed each sample to get a sense of the harmonics present in the sound of the flute.

To do this, I ran each sample through a Fast Fourier Transform (FFT). This converts each sample into the frequency domain, showing, for each frequency contributing to the overall sound, the amplitude of its contribution. The frequency domain is slightly noisy due in part to the background noise in the recording and the imperfections in my own playing, so frequency peaks tend to be spread out over a few frequency bands. I ran the FFT results through a peak finding algorithm to automatically identify, approximately, the peaks in the spectrum. Each is marked with an X in the figures below.

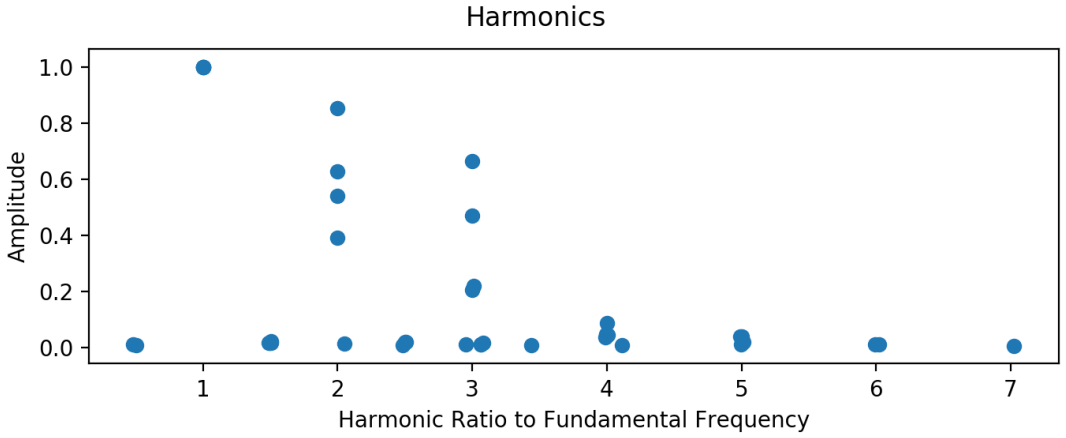








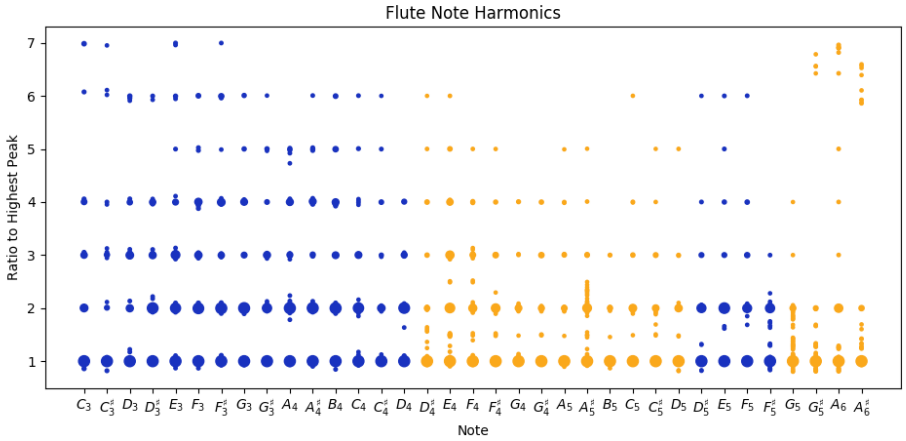
For each peak found, I graphed its amplitude relative to the ratio of its frequency to the frequency of the note I was playing:



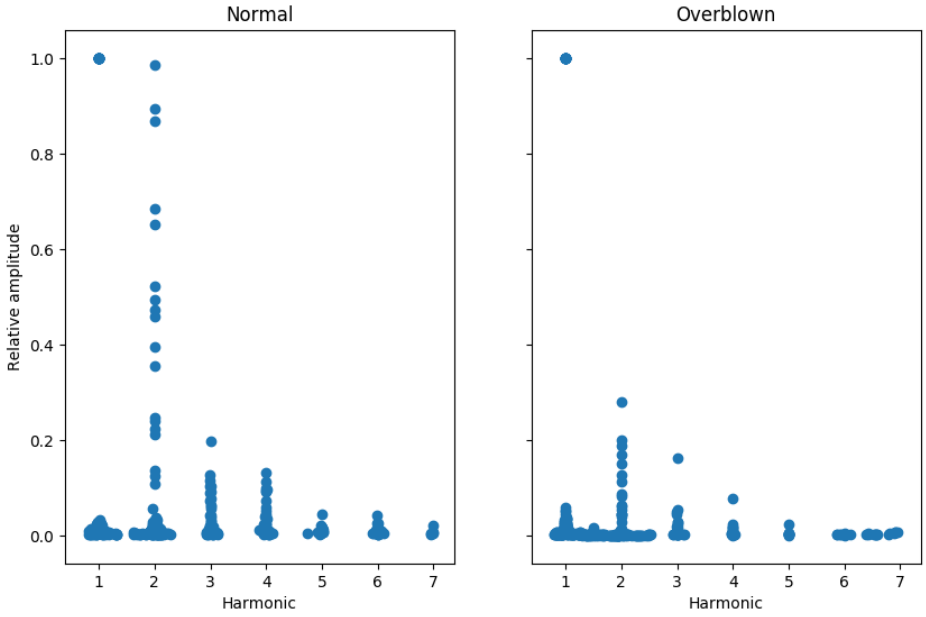
Here we can see that the peaks come in multiples of 0.5. Why does this occur?

All of these frequencies are integer multiples of the fundamental. If the harmonics look like they are all multiples of 0.5, like what we saw in the harmonics from the recordings, the actual fundamental frequency is half of the frequency of the note I was playing. Notes like this are played by overblowing to change the air pressure in the instrument, preventing the first harmonic from sounding.

To figure out which notes this applies to, I played an ascending scale and measured the ratios of the harmonics that sounded to the highest amplitude frequency. Notes for which harmonics come in multiples of 0.5 instead of 1 indicate overblown notes. The chart below shows a dot for each harmonic picked up by the peak finder on the FFT output, with dot size corresponding to relative amplitude. The overblown notes have been highlighted orange.



Here is another plot of the relative amplitude of harmonics, segmented into the overblown and normal samples:



The large spread of low amplitude values is largely due to noise surrounding the real peak that got picked up by the peak detector. Even in the clear peak regions, there is large variance. I'm choosing to average the values into a plausible common amplitude that will sound close enough. The amplitudes I picked for each harmonic in additive synthesis are shown in Table 2.

Table 2: Harmonic amplitudes

Harmonic	Normal amplitude	Overblown amplitude
1	1	1
1.5	0	0.05

Harmonic	Normal amplitude	Overblown amplitude
2	0.6	0.3
2.5	0	0.03
3	0.2	0.15
4	0.15	0.1
5	0.05	0.05
6	0.05	0.02
7	0.03	0.02

Noise component

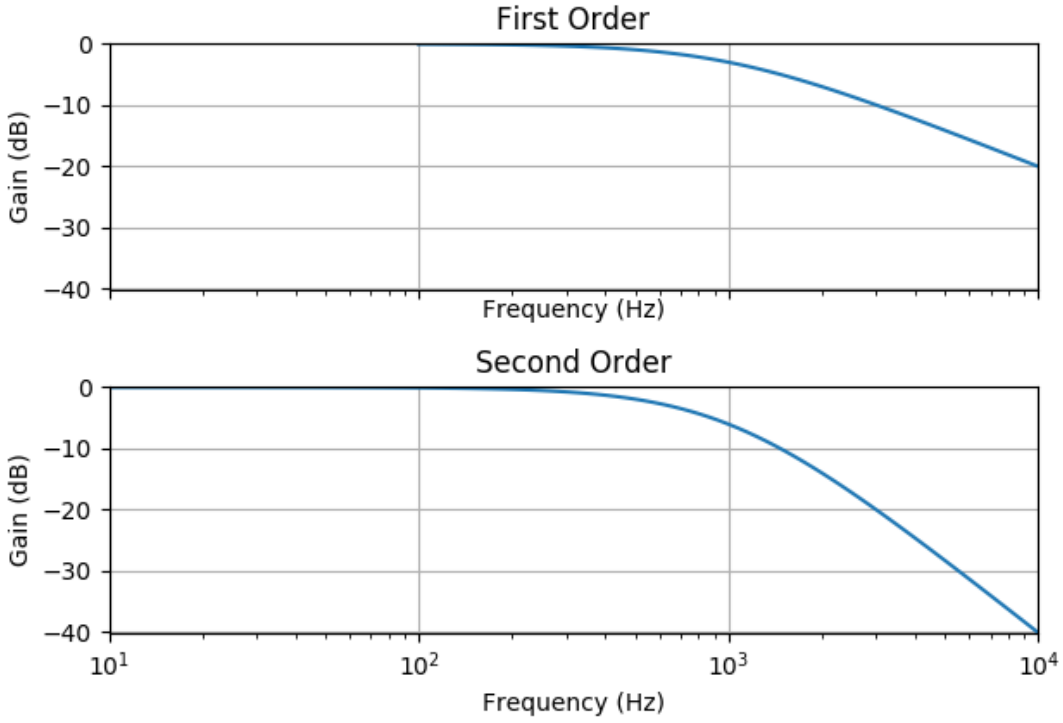
A spectrum analysis of recorded notes shows that there is a noise component to the sound. This low-energy noise is the wind sound from blowing into the instrument, picked up in the recording due to the close proximity of the microphone to the instrument. It makes the flute sound more intimate, so it is a sound I want to replicate in the synthesizer.

Looking at the spectrum produced by the recorded noise component, it is clear that frequencies do not all equally contribute to the noise, as the lower frequencies have higher magnitudes. This tells us that it is not *white noise*, which is defined to be noise where frequency contributions are the same within the region of interest. One way to generate the sort of noise seen in the recording is to use subtractive synthesis, where one starts with white noise, but then quiets down the higher frequencies by applying a filter.

White noise is defined by having equal contributions from each frequency, but there are multiple functions that generate discrete samples that create white noise. One could implement *Gaussian white noise*, where samples have a normal distribution with zero mean. However, Gaussian distributions have an unbounded range, but we need amplitudes to stay between -1 and 1. Instead, we can use a uniform distribution:

```
def noise():
    return np.random.uniform(-1, 1)
```

We then need to filter down this noise using some kind of lowpass filter, that leaves low frequencies the same, but reduces the importance of higher frequencies. A simple first-order system could work, but for frequencies above the bandwidth frequency ω_{BW}, high frequencies roll off at -20dB per decade. In order to have high frequencies roll off faster, higher order systems are required. A second-order system rolls off at -40dB per decade, shown below.



The steeper rolloff rate more closely matches the recorded data, so I decided to implement a second order low pass filter. Transfer functions for second order systems typically have the form $H(s) = \frac{K\omega_n^2}{s^2+2\zeta\omega_n s+\omega_n^2}$. $\omega_n \approx \omega_{BW}$, which is the cutoff frequency after which we see rolloff. We do not need any extra gain, so set the gain $K = 1$. ζ dictates the oscillatory behaviour of the system: if it is greater than 1, the system is overdamped, and the filter can overshoot and oscillate when reaching to "catch up" with an input signal. We do not want this to happen, and when $\zeta \leq 1$, the system will not oscillate. We will pick $\zeta = 1$ so that the system is critically damped and will reach its equilibrium point as fast as it can without oscillating.

To actually implement this as a digital filter, we need to turn the transfer function into a discrete time equation relative to the past discrete samples.

$$
\begin{aligned}
H(s) &= \frac{\omega_{BW}}{s^2 + 2\omega_{BW}s + \omega_{BW}^2} \\
\frac{Y(s)}{X(s)} &= \frac{1}{\frac{s^2}{\omega_{BW}^2} + \frac{2s}{\omega_{BW}} + 1} \\
X(s) &= Y(s)\left(\frac{s^2}{\omega_{BW}^2} + \frac{2s}{\omega_{BW}} + 1\right) \\
X(s) &= \frac{s^2}{\omega_{BW}^2}Y(s) + \frac{2s}{\omega_{BW}}Y(s) + Y(s)
\end{aligned}
$$

Having separated input from output, we can bring the equation into the time domain using the inverse Laplace transform, assuming $y(0) = \left.\frac{dy(t)}{dt}\right|_{t=0} = \left.\frac{d^2y(t)}{dt^2}\right|_{t=0} = 0$:

$$
\begin{aligned}
\mathcal{L}^{-1}\{U(s)\} &= \mathcal{L}^{-1}\left\{\frac{s^2}{\omega_{BW}^2}Y(s) + \frac{2s}{\omega_{BW}}Y(s) + Y(s)\right\} \\
x(t) &= \frac{1}{\omega_{BW}^2}\left(\frac{d^2y(t)}{dt^2}\right) + \frac{2}{\omega_{BW}}\left(\frac{dy(t)}{dt}\right) + y(t)
\end{aligned}
$$

We can then bring everything into discrete time by approximating first and second derivatives and rearranging for $y(t)$:

$$x(t) = \frac{1}{\omega_{BW}^2}\left(\frac{y(t) - 2y(t-\Delta t) + y(t-2\Delta t)}{\Delta t^2}\right) + \frac{2}{\omega_{BW}}\left(\frac{y(t) - y(t-\Delta t)}{\Delta t}\right) + y(t)$$

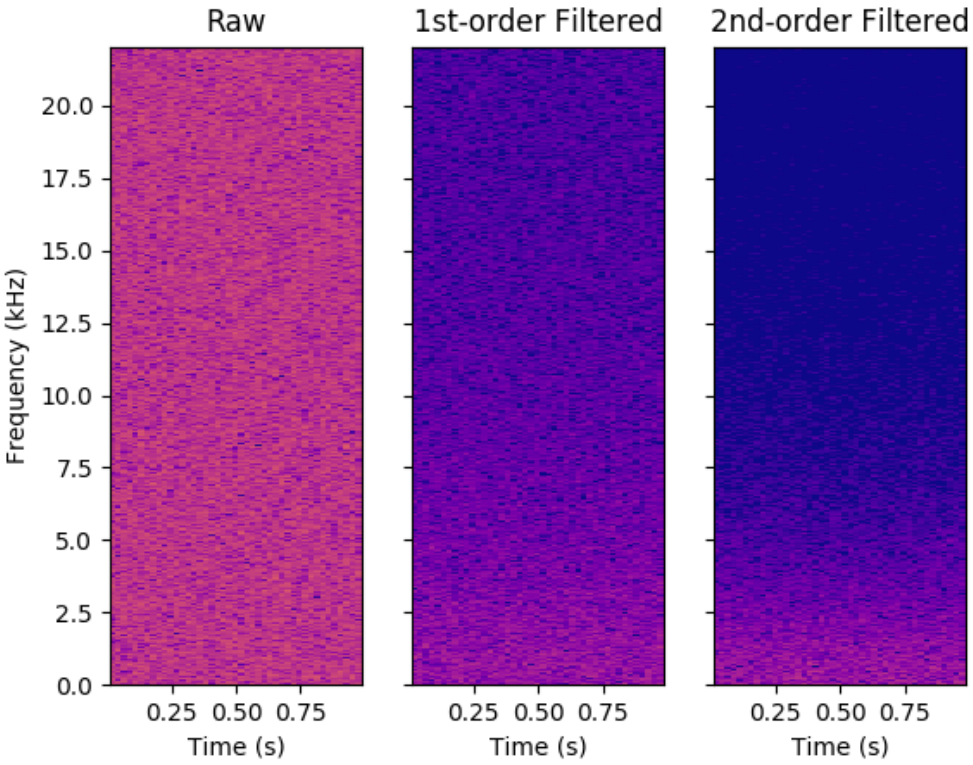
$$y(t) = \frac{x(t) - \left(\frac{-2}{\omega_{BW}\Delta t} - \frac{2}{\omega_{BW}^2\Delta t^2}\right)y(t-\Delta t) - \left(\frac{1}{\omega_{BW}^2\Delta t^2}\right)y(t-2\Delta t)}{1 + \frac{2}{\omega_{BW}\Delta t} + \frac{1}{\omega_{BW}^2\Delta t^2}}$$

$$\begin{aligned} y(t) = &\left(\frac{\omega_{BW}^2\Delta t^2}{\omega_{BW}^2\Delta t^2 + 2\omega_{BW}\Delta t + 1}\right)x(t) \\ &- \left(\frac{-2\omega_{BW}\Delta t - 2}{\omega_{BW}^2\Delta t^2 + 2\omega_{BW}\Delta t + 1}\right)y(t-\Delta t) \\ &- \left(\frac{1}{\omega_{BW}^2\Delta t^2 + 2\omega_{BW}\Delta t + 1}\right)y(t-2\Delta t) \end{aligned}$$

If we let $\alpha = \frac{\omega_{BW}\Delta t}{\omega_{BW}\Delta t + 1}$, then we can see that each new value is effectively a weighted average of the new input and the past two output values:

$$\begin{aligned} y(t) = &\alpha^2 x(t) \\ &- 2(\alpha - 1)y(t-\Delta t) \\ &- (1-\alpha)^2 y(t-2\Delta t) \end{aligned}$$

We can then find the value of α using $\Delta t = 44100$Hz, the sampling frequency, and $\omega_{BW} = 15000$Hz, the approximate frequency around which we want to start seeing rolloff. Here is what white noise looks like, subjected to a first-order lowpass filter and this second-order filter:



Time variance

Results

References

[1] C. R. Nave, “The flute” [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/Music/flute.html

[2] C. R. Nave, “Resonances of open air columns” [Online]. Available: http://hyperphysics.phy-astr.gsu.edu/hbase/Waves/opecol.html

[3] Édouard, “Why are pianos traditionally tuned "out of tune" at the extremes?” 2015 [Online]. Available: https://music.stackexchange.com/questions/14244/why-are-pianos-traditionally-tuned-out-of-tune-at-the-extremes

[4] D. Pagurek, “Throw the dice.” 2018 [Online]. Available: https://soundcloud.com/davidpvm/throw-the-dice